CENG313 **Fall 2018**

PROGRAMMING ASSIGNMENT 3

Due date: 23.11.2018 10:00 am

Write a C program similar to a shell.

Your shell-like program displays the command line prompt “myshell>” and waits for the user's command. It reads the user’s next command, parses it into separate tokens that are used to fill the argument vector for the command to be executed, and executes it.

Your shell-like program should support the following built-in commands (which are not part of the regular shell, the functionality is built directly into your shell itself):

1) cd <directory> : change the current directory to <directory>

- You simply call chdir() function.
- If the <directory> argument is not present, your shell should change the working directory to the path stored in the $HOME environment variable. You can use getenv("HOME") to obtain this.
- You need to support relative and absolute paths.
- The command should also change the PWD environment variable.

2) dir : print the current working directory

- You simply call getcwd() function.

3) history : print 10 most recently entered commands in your shell

- You are not allowed to use “history” Linux command.
- You need to maintain a FIFO list as your data structure.
- Note that the commands will be listed in the issue order where each line displays a number (from 1 to 10) followed by the command issued.

4) findloc <command> : print the location of the executable file corresponding to the given command

- You should use the PATH environment variable as a list of directories in which to look for the command. If the PATH environment variable has the following,
  PATH = /usr/bin:/usr/local/bin/:.
  Then, it will first look in the current directory, then in /usr/local/bin, and finally in /usr/bin until it finds the location of the executable.
- You need to find a file with the right name and the file must be executable.
- As an example, if you type: “findloc ls”, your shell should print out the full path to the ls command which is /usr/bin/ls.
- If the corresponding file does not exist or it is not an executable file, your shell should display an error message.

5) bye : terminate your shell process.

- You simply call exit(0) in your C program.

For any other commands, your shell-like program should consider them as system commands. For system commands, your program creates a child process using fork system call, and the child process executes the command by using execvp() function (You can assume that the command does not include a pathname).

- You need to handle both foreground and background processes for system commands. When a process runs in foreground, your program should wait for the task to complete, then immediately prompt the user for another command. A background process is indicated by placing an ampersand (’&’) character at the end of an input line. When a process run in background, your program should not wait for the task to complete, but immediately prompt the user for another command.
- You are not allowed to use system() function.

Your shell-like program should support pipe operator between two processes.

- For a pipe in the command line, you need to connect stdout of the left command to stdin of the command following the "|". For example, if the user types "ls -al | sort", then the “ls” command is run with stdout directed to a Unix pipe, and that the sort command is run with stdin coming from that same pipe.
- You don't need to support multiple pipes.

The outline of your program should be similar to this:

```
while(1)
        print "myshell>"
        read command line
        parse command
        if the command is built-in
                execute command //in your implementation
        else
                fork a child
                if child
                        execute command //calls execvp
                else if not background process
                        wait for the child
```

Example execution:

```
./myshell
myshell>dir
/home/std/Desktop
myshell>cd /home/std
myshell>dir
/home/std
myshell>findloc ls
/usr/bin/ls
myshell>history
[1] dir
[2] cd /home/std
[3] dir
[4] findloc ls
[5] history
myshell>ls -l
```

```
-rw------- 1 std std 152144 Jun 20 2005 alice-in-wonderland.txt
-rw------- 1 std std 13421 Jun 20 2005 calaveras-county.txt
-rw------- 1 std std 635 Jun 20 2005 french.txt
-rw------- 1 std std 172541 Jun 20 2005 looking-glass.txt
drwx------ 14 std std 476 May 25 2007 shakespeare
myshell>gedit &
myshell>ls -l | wc -l
5
myshell>bye
```

Notes:

- Use Linux man pages for any function that you want to learn about (For example, if you want to learn about chdir function, type “man chdir” in your Linux terminal).
- You can assume that a command line input will contain at most 100 characters and at most 10 arguments.
- You can assume that all command line arguments will be delimited from other command line arguments by white space.
- Consider all necessary error checking for the programs.
- You can work in a 2-member group.
- No late homework will be accepted.

Submission: You are required to submit your **extensively commented** source code to CMS. Please create a compressed file including all source files; and name it as yourstudentnumber(s)_P3.zip (e.g. If your student number is 201812345678, the file name must be 201812345678_P3.zip; if you have a teammate with student number 201812345679, the file name must be 201812345678_201812345679 _P3.zip).